





Este material pode ser utilizado livremente respeitando-se a licença Creative Commons: Atribuição – Uso Não Comercial – Compartilhamento pela mesma Licença (by-nc-sa).



[Ver o Resumo da Licença](#) | [Ver o Texto Legal](#)



# Organização

## Introdução

Histórico

Pré-requisitos

## JUnit Básico

Instalação

O Framework

## O Exemplo

Compilando e Executando via Linha de Comando

Projeto Maven no Eclipse

Copiando a Aplicação para o Projeto

Entendendo o Código

Criando Casos de Teste

Encontrando uma Falha e Corrigindo o Defeito

Métodos Especiais

## JUnit Avançado

Temporização em Casos de Teste

Teste de Exceções

Definindo um Conjunto de Teste

Executando os Teste sem o Eclipse

Ignorando Casos de Teste

Testes Parametrizados



## Introdução

- Histórico
- Pré-requisitos

## JUnit Básico

- Instalação
- O Framework

## O Exemplo

- Compilando e Executando via Linha de Comando
- Projeto Maven no Eclipse
- Copiando a Aplicação para o Projeto
- Entendendo o Código
- Criando Casos de Teste
- Encontrando uma Falha e Corrigindo o Defeito
- Métodos Especiais

## JUnit Avançado

- Temporização em Casos de Teste
- Teste de Exceções
- Definindo um Conjunto de Teste
- Executando os Teste sem o Eclipse
- Ignorando Casos de Teste
- Testes Parametrizados

## Ferramentas Similares

- Referências



## Histórico

- ▶ Desenvolvido por Kent Beck e Erich Gamma, o framework de teste de unidade JUnit se tornou uma das bibliotecas Java mais utilizadas no mundo.
- ▶ De acordo com Martin Fowler “Never in the field of software development was so much owed by so many to so few lines of code”.
- ▶ Início do JUnit: 1994.
- ▶ Versão 3.8: lançada em agosto de 2002.
- ▶ Versão 3.8.1: lançada em setembro de 2002 com defeitos corrigidos.
- ▶ Versão 4.0: lançada em fevereiro de 2006.
- ▶ Versão 3.8.2: lançada em março de 2006.
- ▶ ...
- ▶ Versão 4.11: lançada em novembro de 2012.



## Pré-requisitos

## Pré-requisitos

- ▶ Conhecimento da linguagem Java.
- ▶ Conhecimento de teste de software.
- ▶ Kit de desenvolvimento Java versão 1.5 ou superior. Versão 4.x do JUnit utiliza recursos de anotações disponível apenas a partir desta versão do Java.
- ▶ Desejável: conhecimento de Eclipse e Maven.

Introdução

JUnit Básico

## Instalação

## O Framework



## Instalação

## Instalação (1)

- ▶ Página do projeto: <http://www.junit.org/>
- ▶ Distribuído em forma compactada.
- ▶ A versão 4.11 traz uma série de melhorias em relação a versão predecessora 3.8.2.

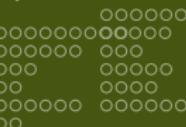


## Instalação

## Instalação (2)

- ▶ Para usar a ferramenta, basta fazer o download dos arquivos jars abaixo e incluí-los no CLASSPATH do projeto.
  - ▶ junit.jar – [junit-4.11.jar](#)
  - ▶ hamcrest-core.jar – [hamcrest-core-1.3.jar](#)
- ▶ Opcionalmente, basta criar os projetos usando Maven e adicionar a dependência abaixo no arquivo pom.xml.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```



## O Framework

# O Framework JUnit

- ▶ Framework de código aberto para o teste de programas Java.
- ▶ Passos básicos para uso do framework:
  - ▶ Criar alguns objetos.
  - ▶ Enviar algumas mensagens a esses objetos.
  - ▶ Verificar se o resultado obtido é igual ao esperado com o uso de asserções (disponíveis no framework).

Introdução

## O Exemplo

## Compilando e Executando via Linha de Comando

Projeto Maven no Eclipse

### Copiando a Aplicação para o Projeto

## Entendendo o Código

## Criando Casos de Teste

## Encontrando uma Falha e Corrigindo o Defeito

Métodos Especiais



Compilando e Executando via Linha de Comando

## A Especificação Exemplo

- ▶ Programa Identifier: parte de um compilador que verifica se um identificador é válido em dada linguagem de programação
- ▶ Código fonte: Identifier.java e IdentifierMain.java (disponíveis no material do treinamento)
- ▶ Classe principal: IdentifierMain
- ▶ Forma de execução:
  - ▶ Identificador deve ser fornecido como parâmetro na invocação do programa.



Compilando e Executando via Linha de Comando

## Compilando o Identifier

```
cd src
```

```
src$ javac Identifier.java IdentifierMain.java
```



Compilando e Executando via Linha de Comando

## Executando o Identifier

Para invocar o programa, basta executar o comando abaixo, dentro do diretório onde o programa foi instalado:

```
1 cd src
2
3 src$ java IdentifierMain
4 Uso: IdentifierMain <string>
5
6 src$ java IdentifierMain "abc12"
7 Valido
8
9 src$ java IdentifierMain "cont*1"
10 Invalido
11
12 src$ java IdentifierMain "1soma"
13 Invalido
14
```



## Projeto Maven no Eclipse

## Invocando o Eclipse

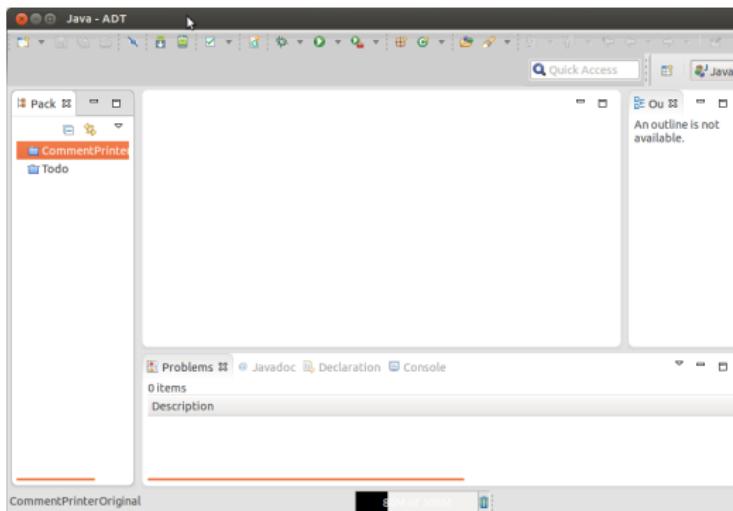
- ▶ Durante o treinamento, os exemplos serão executados, preferencialmente, no Eclipse.
- ▶ A versão utilizada é a que acompanha o <http://developer.android.com/sdk/index.html>, já configurada com a maioria dos plugins necessários para o treinamento, incluindo o JUnit.
- ▶ Plug-ins e ferramentas adicionais serão instalados quando necessário.
- ▶ Para invocar o Eclipse, basta executar:

```
<DIRETORIO_INSTALACAO>/adt-bundle-linux-x86_64-20131030/eclipse/eclipse
```



## Projeto Maven no Eclipse

## Tela Inicial do Eclipse





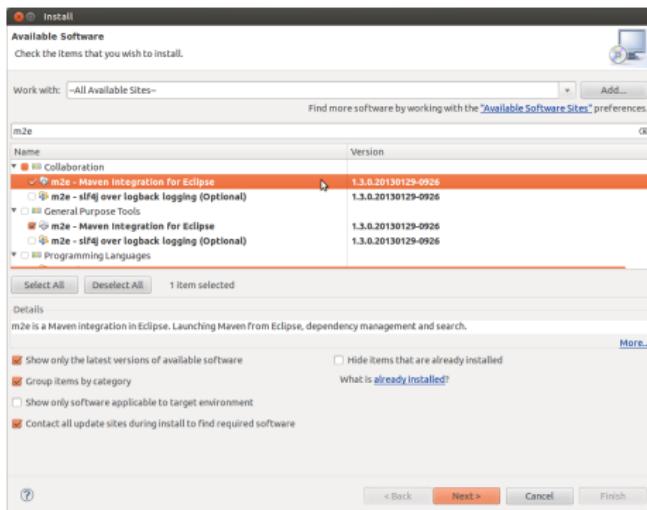
## Instalando o Plugin m2eclipse

- ▶ Aberto o Eclipse, entre no menu **Help->Install New Software...**
  - ▶ No campo **Work with** selecione – **All Available Sites** –
  - ▶ No filtro preencha com **m2e** (abreviatura para Maven To Eclipse plugin)
  - ▶ Após aparecerem as opções, selecione para instalação o **m2e - Maven Integration for Eclipse**
  - ▶ Prossiga com a instalação normal. Ao final do processo, reinicialize o Eclipse para concluir a instalação.



## Projeto Maven no Eclipse

## Tela Instalação de Plugins

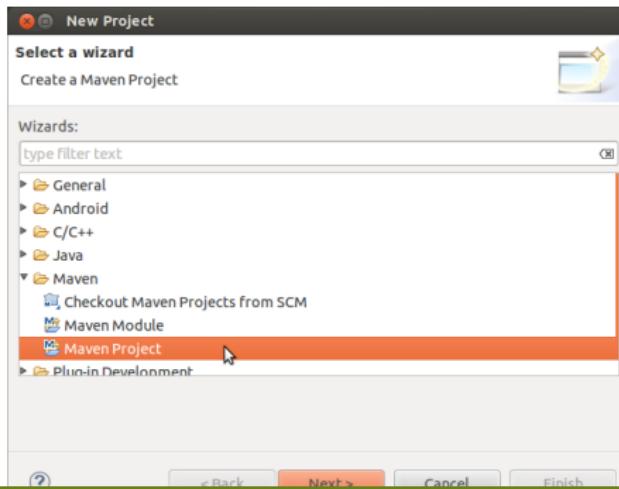




## Projeto Maven no Eclipse

## Criando um Projeto Maven (1)

- ▶ Aberto o Eclipse, entre no menu **File->New->Project...**
- ▶ Escolha **Maven** e **Maven Project**

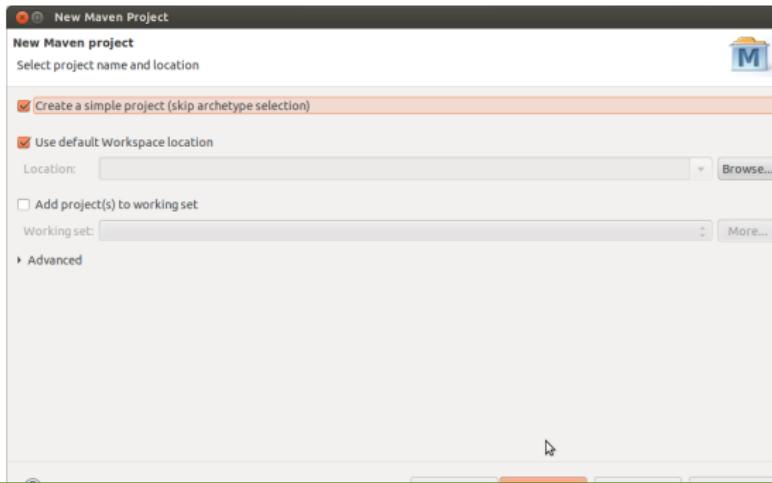




## Projeto Maven no Eclipse

## Criando um Projeto Maven (2)

- ▶ Em seguida, assinale a opção **Create simple project (skip archetype selection)**





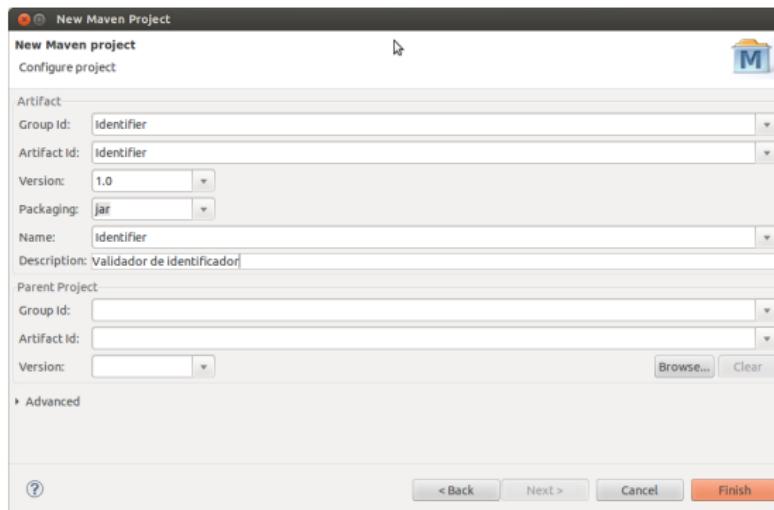
## Criando um Projeto Maven (3)

- Em seguida, preencha os campos para o projeto Identifier, conforme tela a seguir e finalmente clique em **Finalizar**.



## Projeto Maven no Eclipse

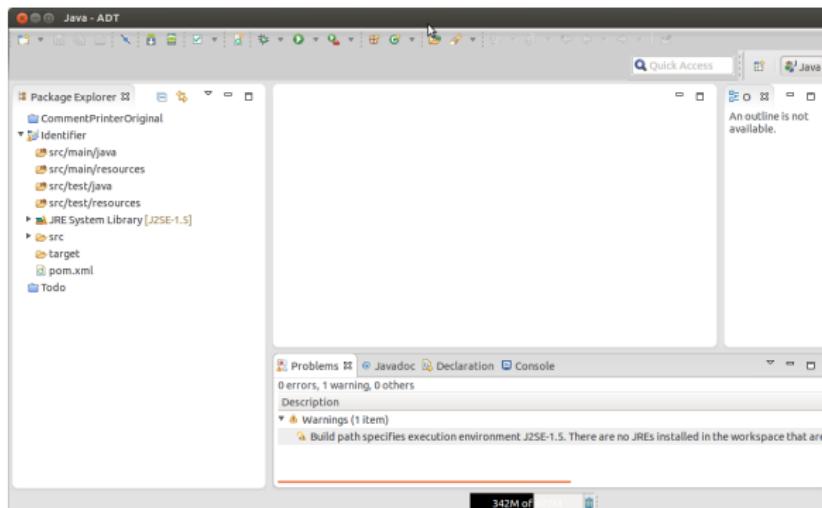
## Criando um Projeto Maven (4)





## Projeto Maven no Eclipse

## Criando um Projeto Maven (5)





## Projeto Maven no Eclipse

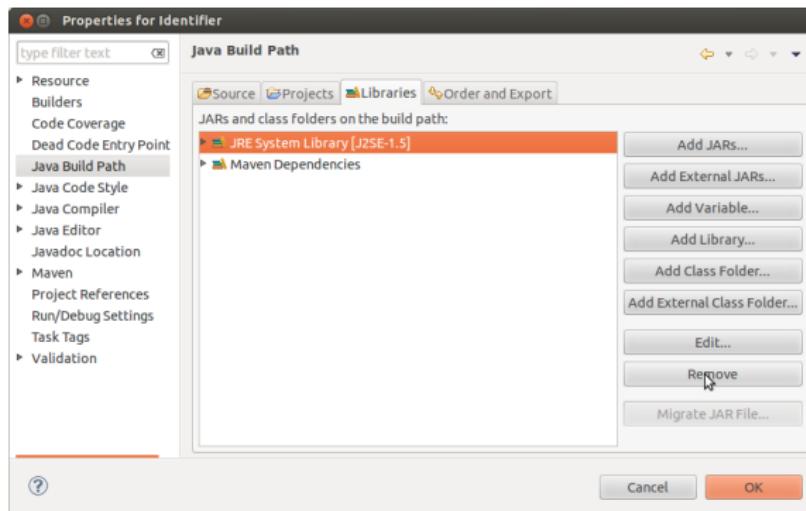
## Criando um Projeto Maven (6)

- ▶ Observe na tela anterior que o Eclipse está acusando um warning nesse projeto.
- ▶ Ambiente de execução do J2SE-1.5 definido no projeto, mas essa versão do Java não está disponível.
- ▶ Para alterar isso, clique com o direito do mouse sobre o nome do projeto `Identifier` e escolha a última opção do menu suspenso **Properties...**



## Projeto Maven no Eclipse

## Criando um Projeto Maven (7)





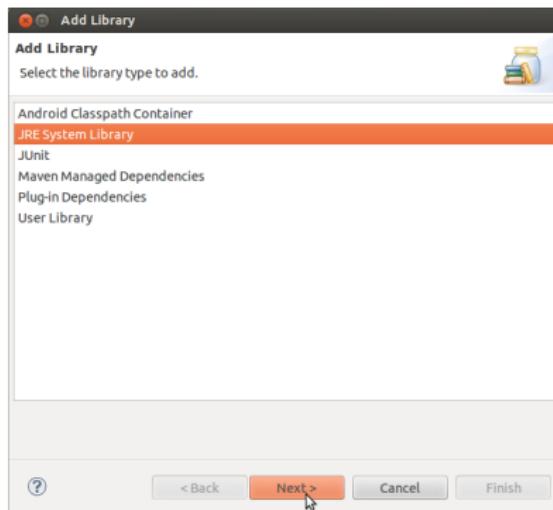
## Criando um Projeto Maven (8)

- ▶ Na tela que abrir, selecione **JRE System Library [J2SE-1.5]** e clique em remover.
- ▶ Em seguida clique no botão **Add Library...** e selecione **JRE System Library** e clique em **Next**



## Projeto Maven no Eclipse

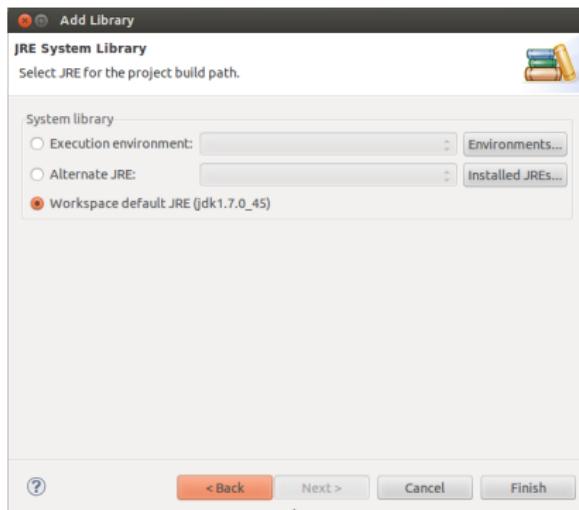
## Criando um Projeto Maven (9)





## Projeto Maven no Eclipse

## Criando um Projeto Maven (10)





Projeto Maven no Eclipse

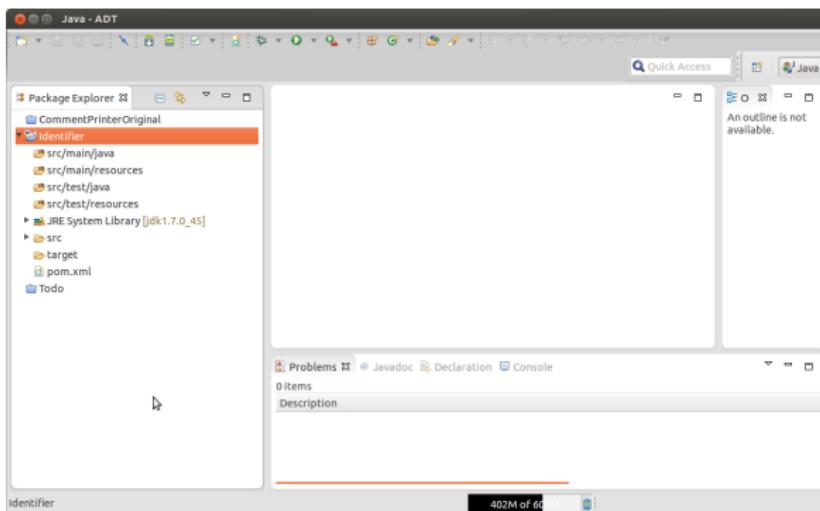
## Criando um Projeto Maven (11)

- ▶ Escolha a opção padrão e clique em **Finish**
- ▶ Finalmente, aparecerá o projeto criado sem warnings associado.



## Projeto Maven no Eclipse

## Criando um Projeto Maven (12)





## Copiando a Aplicação para o Projeto

# Copiando a Aplicação para o Projeto

- ▶ Copie os arquivos Identifier.java e IdentifierMain.java para dentro da pasta src/main/java.
- ▶ Após copiar, clique com o botão direito sobre o nome do projeto Identifier e escolha a opção **Refresh** do menu suspenso.
- ▶ Com isso, o Eclipse irá compilar o projeto e, ao final, a aplicação está pronta para ser executada de dentro do Eclipse.



Copiando a Aplicação para o Projeto

## O Método Principal (1)

- ▶ O método principal do programa Identifier é o método:

```
public boolean validateIdentifier(String s)
```

- ▶ Recebe um string s que corresponde a um dado identificador
- ▶ Retorna true se for um identificador válido em **Silly Pascal**, ou false caso contrário.

*“Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo um e no máximo seis caracteres de comprimento.”*



Copiando a Aplicação para o Projeto

## O Método Principal (2)

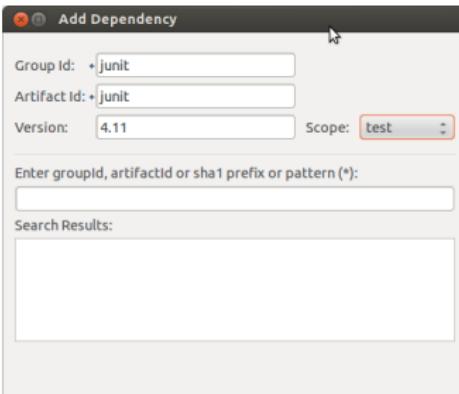
```
2  public boolean validateIdentifier(String s) {
3      char achar;
4      boolean valid_id = false;
5      achar = s.charAt(0);
6      valid_id = valid_s(achar);
7      if (s.length() > 1) {
8          achar = s.charAt(1);
9          int i = 1;
10         while (i < s.length() - 1) {
11             achar = s.charAt(i);
12             if (!valid_f(achar))
13                 valid_id = false;
14             i++;
15         }
16     }
17
18     if (valid_id && (s.length() >= 1) && (s.length() < 6))
19         return true;
20     else
21         return false;
```



Copiando a Aplicação para o Projeto

## Incluindo Dependência do JUnit no Maven (1)

- ▶ Antes de iniciar a criação da classe de teste é possível incluir a dependência da biblioteca do JUnit no projeto Maven.
  1. Clique o botão direito do mouse sobre projeto Identifier.
  2. Escolha a opção **Maven->Add Dependency**. Preencha a tela conforme abaixo:





Copiando a Aplicação para o Projeto

## Incluindo Dependência do JUnit no Maven (2)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>Identifier</groupId>
  <artifactId>Identifier</artifactId>
  <version>1.0</version>
  <name>Identifier</name>
  <description>Validador de identificador</description>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



Copiando a Aplicação para o Projeto

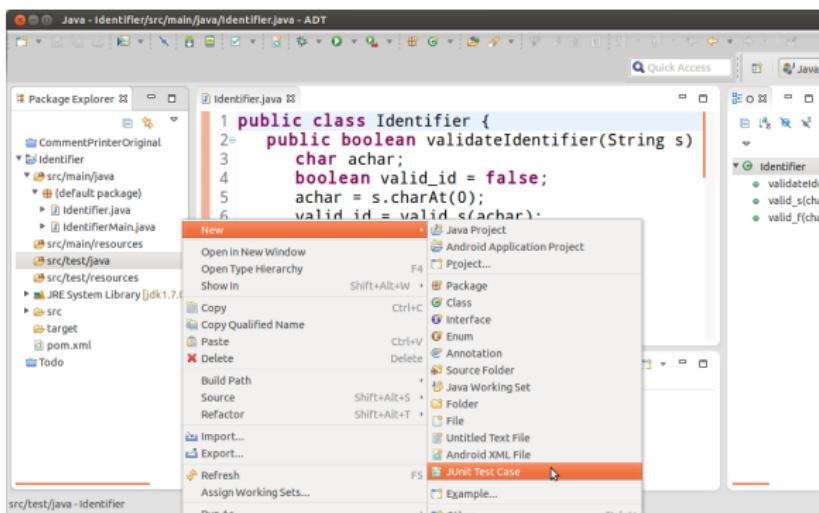
## Criando o Template do JUnit (1)

- ▶ Clique com o botão direito sobre a pasta `src/test/java` no projeto `Identifier`.
- ▶ Escolha a opção **New->JUnit Test Case**.



Copiando a Aplicação para o Projeto

## Criando o Template do JUnit (2)





Copiando a Aplicação para o Projeto

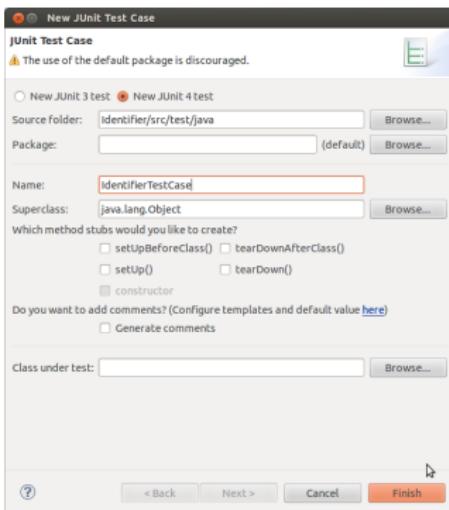
## Criando o Template do JUnit (3)

- ▶ Na janela que abrirá, preencha apenas o nome da classe de teste a ser criada com `IdentifierTestCase` (sem a extensão) e clique **Finish**
- ▶ Veja tela a seguir para detalhes.



Copiando a Aplicação para o Projeto

## Criando o Template do JUnit (4)





Copiando a Aplicação para o Projeto

## Criando o Template do JUnit (5)

- ▶ O código criado é bastante simples e não dá ideia do poder que existe por traz dele.

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Test;
4
5
6 public class IdentifierTestCase {
7
8     @Test
9     public void test() {
10         fail("Not yet implemented");
11     }
12 }
13 }
```



## Entendendo o Código

## Entendendo o Código (1)

- ▶ Uma classe de teste do JUnit pode conter vários casos de teste.
- ▶ Um caso de teste é um método dentro dessa classe, precedido da anotação `@Test`.
- ▶ É uma boa prática de escrita de casos de teste unitário separar cada caso de teste em um método de teste.
- ▶ Da mesma forma, deve-se agrupar em uma classe de teste os casos de teste referentes a determinada classe e/ou método em teste, visando facilitar a localização dos testes.



## Entendendo o Código

## Entendendo o Código (2)

- ▶ A primeira parte do código inclui as classes exigidas pelo framework até o momento.
- ▶ Classes de asserções do pacote `org.junit.Assert` que implementam os métodos que permitem a comparação entre o resultado obtido e o esperado.
- ▶ Classe `org.junit.Test` que implementa a anotação `@Teste`.

```
1 import static org.junit.Assert.*;  
2  
3 import org.junit.Test;
```



## Entendendo o Código

## Entendendo o Código (3)

- ▶ Em seguida vem a definição do nome da classe de teste `IdentifierTestCase` que contém, até o momento, um único método de teste, denominado `test()`.
- ▶ Na classe de teste, podem existir tantos métodos quanto necessário mas apenas aqueles precedidos da anotação `@Test` é que são considerados pelo framework como sendo casos de teste.

```
6  public class IdentifierTestCase {  
7  
8      @Test  
9      public void test() {  
10          fail("Not yet implemented");  
11      }  
12  }
```



## Entendendo o Código

## Entendendo o Código (4)

- ▶ Um método de teste sempre deve:
  - ▶ ter modificador de acesso `public`
  - ▶ ter retorno do tipo `void`
  - ▶ não possuir parâmetros

```
8  @Test
9  public void test() {
10     fail("Not yet implemented");
11 }
```



## Entendendo o Código

## Entendendo o Código (5)

- ▶ No corpo de um método de teste basicamente o que se faz é:
  1. criar um objeto da classe que contém o método a ser testado (se o método a ser testado for um método de instância)
  2. invocar o método em teste com os parâmetros desejados e armazenar o valor de retorno
  3. utilizar uma das asserções disponíveis no framework para comparar o resultado obtido com o resultado esperado.
- ▶ No template criado, nenhum dos passos acima é realizado.
- ▶ O método contém apenas uma chamada ao método `fail`, disponível no pacote `org.junit.Assert`, que, se executado, acusa que o teste falhou.

8

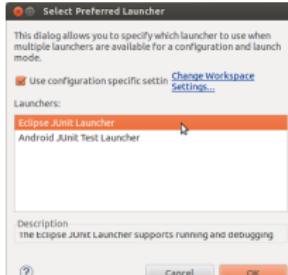
```
@Test  
public void test() {
```



## Entendendo o Código

## Entendendo o Código (6)

- ▶ Para executar o teste, basta clicar com o botão direito sobre o nome do arquivo de casos de teste e escolher a opção **Run As->JUnit Test**
- ▶ Na primeira execução, é solicitado ao usuário escolher qual executor de teste é para ser usado. Escolha o **Eclipse JUnit Launcher** e clique em **Ok**.





## Entendendo o Código

## Entendendo o Código (7)

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Shows the title "Java - Identifier/src/test/java/IdentifierTestCase.java - ADT".
- Left Sidebar:** Displays the package structure under "IdentifierTestCase". It shows one file, "IdentifierTestCases.java", with the status "Runs: 1/1 Errors: 0 Failures: 0".
- Central Editor:** Shows the Java code for "IdentifierTestCases.java":

```
1+import static org.junit.Assert.*;
2
3
4
5 public class IdentifierTestCases {
6
7     @Test
8     public void test() {
9         fail("Not yet implemented");
10    }
11
12
13 }
14
```
- Right Panel:** A JUnit test runner interface titled "Assert.Assert()<implicit constructor>". It has a table with one row labeled "Test Method".
- Bottom Panels:** Includes "Failure Trace", "Problems", "Javadoc", "Declaration", and "Console". The "Console" panel shows the output: "<terminated> IdentifierTestCases [JUnit] /local/tools/jdk1.7.0\_45/bin/java (28/01/2014 02:01:15)".



## Criando Casos de Teste

## Criando Casos de Teste (1)

- ▶ Para a criação de um caso de teste é necessário:
  - ▶ Identificar o método a ser testado: `validateIdentifier`
  - ▶ Compreender a especificação do método: o que recebe de entrada e qual a saída produzida em função da entrada escolhida.
    - ▶ Entrada: cadeia de caracteres (`String`)
    - ▶ Saída: `true` ou `false`
  - ▶ Comparar a saída produzida (obtida) com aquela que deveria ser gerada conforme a especificação.



## Criando Casos de Teste

## Criando Casos de Teste (2)

- ▶ Primeiro teste completo.

```
7  @Test
8  public void validate01() {
9      Identifier id = new Identifier();
10     boolean obtido;
11     obtido = id.validateIdentifier("al");
12     assertEquals(true, obtido);
13 }
```



## Criando Casos de Teste

## Criando Casos de Teste (3)

- ▶ Código completo da classe de teste:

IdentifierTestCase.java, conforme JUnit  
Versão 4.11.

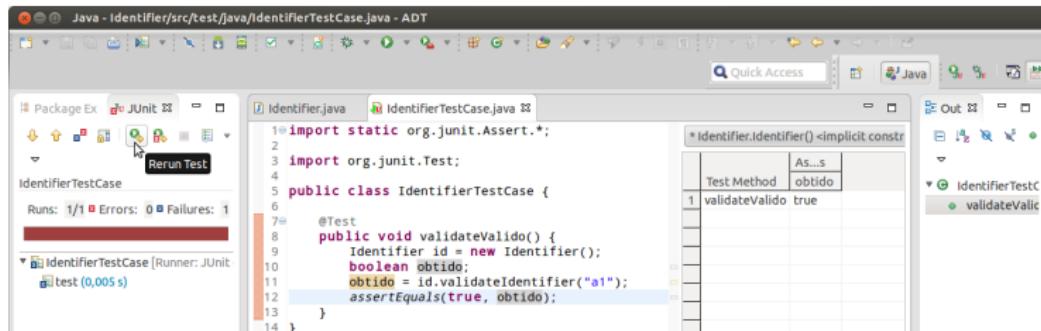
```
1 import static org.junit.Assert.*;
2
3 import org.junit.Test;
4
5 public class IdentifierTestCase {
6
7     @Test
8     public void validate01() {
9         Identifier id = new Identifier();
10        boolean obtido;
11        obtido = id.validateIdentifier("a1");
12        assertEquals(true, obtido);
13    }
}
```



## Criando Casos de Teste

## Criando Casos de Teste (4)

- ▶ Novamente, para executar o teste criado basta clicar com o botão direito sobre o nome do arquivo de casos de teste e escolher a opção **Run As->JUnit Test**; ou
- ▶ Se a janela do JUnit continua aberta, basta clicar no ícone para reexecutar o conjunto de teste, conforme abaixo.





## Criando Casos de Teste

## Criando Casos de Teste (4)

- ▶ Resultado final: teste aprovado.
- ▶ Ou seja, para a entrada "a1" que representa, conforme a especificação um identificador válido, o método validateIdentifier retornou true.
- ▶ O método assertEquals compara o resultado esperado true com o obtido, armazenado na variável obtido que, nesse caso, também continha o valor true.

```
Java - Identifier/src/test/java/IdentifierTestCase.java - ADT
Java - Identifier/src/test/java/IdentifierTestCase.java - ADT
Identifier.java IdentifierTestCase.java
1 import static org.junit.Assert.*;
2
3 import org.junit.Test;
4
5 public class IdentifierTestCase {
6
7     @Test
8     public void validateValido() {
9         Identifier id = new Identifier();
10        boolean obtido = id.validate("a1");
11        assertEquals(true, obtido);
12    }
13 }
```



## Criando Casos de Teste

## Criando Casos de Teste (5)

- ▶ Isso é tudo o que se precisa saber no momento para se criar e executar de forma automática teste com o JUnit.
- ▶ Obviamente existe muitos outros recursos que serão explorados durante o treinamento.



Vamos praticar?!



## Encontrando uma Falha e Corrigindo o Defeito

## Encontrando uma Falha (1)

- ▶ Melhorar o conjunto de teste incluindo novos casos de teste

## Conjunto de teste inicial

```
T0 = { ("a1", Valido), ("", Invalido),  
        ("A1b2C3d", Invalido), ("2B3", Invalido),  
        ("Z#12", Invalido) }
```



## Encontrando uma Falha e Corrigindo o Defeito

## Encontrando uma Falha (2)

```
15  @Test
16  public void validate02() {
17      Identifier id = new Identifier();
18      boolean obtido;
19      obtido = id.validateIdentifier("");
20      assertEquals(false, obtido);
21  }
22
23  @Test
24  public void validate03() {
25      Identifier id = new Identifier();
26      boolean obtido;
27      obtido = id.validateIdentifier("A1b2C3d");
28      assertEquals(false, obtido);
29  }
```



## Encontrando uma Falha e Corrigindo o Defeito

## Encontrando uma Falha (3)

```
31     @Test
32     public void validate04() {
33         Identifier id = new Identifier();
34         boolean obtido;
35         obtido = id.validateIdentifier("2B3");
36         assertEquals(false, obtido);
37     }
38
39     @Test
40     public void validate05() {
41         Identifier id = new Identifier();
42         boolean obtido;
43         obtido = id.validateIdentifier("Z#12");
44         assertEquals(false, obtido);
45     }
```



## Encontrando uma Falha e Corrigindo o Defeito

## Encontrando uma Falha (4)

Executando os novos testes...

The screenshot shows the Eclipse IDE interface with the following details:

- Java - Identifier/src/test/java/IdentifierTestCase.java - ADT**: The title bar.
- Package Explorer**: Shows a package named "java" containing "IdentifierTestCase". Below it, there are five test methods: validate02, validate03, validate04, validate05, and validate01. The status bar indicates "Runs: 5/5", "Errors: 1", and "Failures: 0".
- Identifier.java**: The source code for the Identifier class, which contains two methods: validate02 and validate03.
- IdentifierTestCase.java**: The JUnit test case for Identifier. It contains five test methods: validate02, validate03, validate04, validate05, and validate01. The validate02 method fails with an `AssertionError`.
- Failure Trace**: A stack trace for the failed assertion in validate02. It shows:

```
java.lang.StringIndexOutOfBoundsException: String index o  
at java.lang.String.charAt(String.java:658)  
at Identifier.validateIdentifier(Identifier.java:5)  
at IdentifierTestCase.validate02(IdentifierTestCase.java:19)
```
- Outline**: Shows the structure of the IdentifierTestCase class, listing all its methods.
- Problems**, **Javadoc**, **Declaration**, **Console**: Standard Eclipse toolbars.
- Console**: Shows the output: "<terminated> java [JUnit] /local/tools/jdk1.7.0\_45/bin/java [28/01/2014 23:01:28]".



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○

## Encontrando uma Falha e Corrigindo o Defeito

## Corrigindo o Defeito (1)

```
2  public boolean validateIdentifier(String s) {  
3      char achar;  
4      boolean valid_id = false;  
5      achar = s.charAt(0);  
6      valid_id = valid_s(achar);  
7      if (s.length() > 1) {  
8          achar = s.charAt(1);  
9          int i = 1;  
10         while (i < s.length() - 1) {  
11             achar = s.charAt(i);  
12             if (!valid_f(achar))  
13                 valid_id = false;  
14             i++;  
15         }  
16     }  
17  
18     if (valid_id && (s.length() >= 1) && (s.length() < 6))  
19         return true;  
20     else  
21         return false;  
22 }
```



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○

## Encontrando uma Falha e Corrigindo o Defeito

## Corrigindo o Defeito (2)

```
2  public boolean validateIdentifier(String s) {  
3      char achar;  
4      boolean valid_id = false;  
5      if (s.length() > 0) {  
6          achar = s.charAt(0);  
7          valid_id = valid_s(achar);  
8          if (s.length() > 1) {  
9              achar = s.charAt(1);  
10             int i = 1;  
11             while (i < s.length() - 1) {  
12                 achar = s.charAt(i);  
13                 if (!valid_f(achar))  
14                     valid_id = false;  
15                 i++;  
16             }  
17         }  
18     }  
19     if (valid_id && (s.length() >= 1) && (s.length() < 6))  
20         return true;  
21     else  
22         return false;  
23 }
```



## Encontrando uma Falha e Corrigindo o Defeito

## Corrigindo o Defeito (3)

Reexecutando os testes...

The screenshot shows the Eclipse IDE interface with the following components visible:

- Package Explorer:** Shows a green bar indicating "Finished after 0.009 seconds" with "Runs: 5/5", "Errors: 0", and "Failures: 0".
- Identifier.java:** The source code for the Identifier class. It contains methods validateIdentifier and validateSChar. A red underline is under validateSChar.
- IdentifierTestCase.java:** The test case for Identifier. It includes five test methods: validated1, validated2, validated3, validated4, and validated5. The first three have green checkmarks, while the last two have red error icons.
- Outline:** Shows the class structure with methods validateIdentifier, valid\_s(char), and valid\_f(char).
- Problems:** Shows a red error icon next to validateSChar in Identifier.java.



Encontrando uma Falha e Corrigindo o Defeito

## Definindo Assertivas (1)

- ▶ JUnit oferece diversos tipos de assertivas.
- ▶ Para uma lista completa, consulte a [classe Assert na API do JUnit](#)
- ▶ Algumas mais comuns são:
  - ▶ `assertEquals(<esperado>, <obtido>)`
  - ▶ `assertNotSame(<esperado>, <obtido>)`
  - ▶ `assertSame(<esperado>, <obtido>)`
  - ▶ `assertTrue(<expressão lógica>)`
  - ▶ `assertFalse(<expressão lógica>)`
  - ▶  `assertNull(Object)`
  - ▶  `assertNotNull(Object)`
  - ▶  `fail(String)`



## Encontrando uma Falha e Corrigindo o Defeito

## Definindo Assertivas (2)

- ▶ Oráculo: comparar o resultado obtido com o esperado, viabiliza a execução automática dos casos de teste.
- ▶ Para utilizar:
  - ▶ `import org.junit.Assert;`
  - ▶ **Uso:** `Assert.assertEquals(a, b);`
  - ▶ `import static org.junit.Assert.*;`
  - ▶ **Uso:** `assertEquals(a, b);`



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



●



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



○



## Métodos Especiais

## Anotações Especiais do Framework (1)

- ▶ Existem situações em que, antes da execução de cada caso de teste algumas ações precisam ser executadas.
- ▶ Em outras, após a execução de cada caso de teste é que se deve executar alguma ação.
- ▶ Duas anotações especiais são definidas:
  - ▶ @Before – faz com que o método anotado seja executado **antes** de cada **caso de teste**. Utilizado para colocar o programa num estado conhecido.
  - ▶ @After – faz com que o método anotado seja sempre executado **após** cada **caso de teste**. Utilizado, em geral, para liberar recursos utilizado pelo caso de teste.
- ▶ Para a utilização dessas anotações é necessário importar as



## Métodos Especiais

## Anotações Especiais do Framework (2)

Observe a classe de teste abaixo na qual o objeto `id` é inicializado em cada método de teste.

```
7  @Test
8  public void validate01() {
9      Identifier id = new Identifier();
10     boolean obtido;
11     obtido = id.validateIdentifier("a1");
12     assertEquals(true, obtido);
13 }
14
15 @Test
16 public void validate02() {
17     Identifier id = new Identifier();
18     boolean obtido;
19     obtido = id.validateIdentifier("");
20     assertEquals(false, obtido);
21 }
22
23 @Test
24 public void validate03() {
25     Identifier id = new Identifier();
26     boolean obtido;
27     obtido = id.validateIdentifier("Alb2C3d");
28     assertEquals(false, obtido);
```



## Métodos Especiais

## Anotações Especiais do Framework (3)

- ▶ Essa inicialização pode ser feita dentro de um método anotado com `@Before`

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Before;
4 import org.junit.Test;
5
6 public class IdentifierTestCase {
7
8     private Identifier id;
9
10    @Before
11    public void inicializa() {
12        id = new Identifier();
13    }
14
15    @Test
16    public void validate01() {
17        boolean obtido;
18        obtido = id.validateIdentifier("a1");
```



## Métodos Especiais

## Anotações Especiais do Framework (4)

```
22 @Test
23 public void validate02() {
24     boolean obtido;
25     obtido = id.validateIdentifier("");
26     assertEquals(false, obtido);
27 }
28
29 @Test
30 public void validate03() {
31     boolean obtido;
32     obtido = id.validateIdentifier("A1b2C3d");
33     assertEquals(false, obtido);
34 }
35
36 @Test
37 public void validate04() {
38     boolean obtido;
39     obtido = id.validateIdentifier("2B3");
40     assertEquals(false, obtido);
41 }
42
43 @Test
44 public void validate05() {
45     boolean obtido;
46     obtido = id.validateIdentifier("Z#12");
47     assertEquals(false, obtido);
48 }
```



## Métodos Especiais

## Anotações Especiais do Framework (5)

- ▶ Uma novidade introduzida na versão 4.x é a possibilidade de executar algum método antes e depois o início do conjunto de teste como um todo.
- ▶ Duas anotações especiais são definidas para isso:
  - ▶ `@BeforeClass` – faz com que o método anotado seja executado **antes** de cada **classe de teste**. Útil para alocar recursos computacionalmente “caros” uma única vez.
  - ▶ `@AfterClass` – faz com que o método anotado seja sempre executado **após** cada **classe de teste**. Útil para liberar recursos computacionalmente “caros” uma única vez.
- ▶ Para a utilização dessas anotações é necessário importar as classes `org.junit.BeforeClass` e



## Métodos Especiais

## Anotações Especiais do Framework (6)

Síntese das anotações especiais para inicializar e encerrar conjunto de teste e caso de teste.

<b>@BeforeClass e @AfterClass</b>	<b>@Before e @After</b>
Somente um método por classe pode ser anotado.	Múltiplos métodos podem ser anotados. A ordem de execução é indefinida. Métodos sobrescritos não são executados.
Nomes dos métodos são irrelevantes.	Nomes dos métodos são irrelevantes.
Executa uma única vez por classe de teste.	Executa antes/após cada método de teste.
Método <code>@BeforeClass</code> da superclasse é executado antes do da subclasse. Método <code>@AfterClass</code> da superclasse é executado após o da subclasse.	Métodos <code>@Before</code> da superclasse são executados antes dos da subclasse. Métodos <code>@After</code> da superclasse são executados após os da subclasse.
Método deve ser público <code>public</code> e estático <code>static</code> .	Método deve ser público <code>public</code> e não-estático.
É garantida a execução do método <code>@AfterClass</code> mesmo se um método	É garantida as execuções dos métodos <code>@After</code> mesmo se os métodos <code>@Before</code>

## Introdução

JUnit Avançado

## Temporização em Casos de Teste

## Teste de Exceções

## Definindo um Conjunto de Teste

### Executando os Teste sem o Eclipse

## **Ignorando Casos de Teste**

## Testes Parametrizados



## Temporização em Casos de Teste

## Caso de Teste Temporizado (1)

- ▶ Não presente nas versões anteriores à versão 4.x
- ▶ Casos de testes devem executar rapidamente.
- ▶ Impede que um caso de teste execute indefinidamente em caso de looping.
- ▶ Útil no teste de conexões com servidores ou banco de dados.
  - ▶ Caso o servidor esteja lento ou não respondendo, os casos de testes podem continuar a execução.



## Temporização em Casos de Teste

## Caso de Teste Temporizado (2)

Considere a existência de um defeito na linha 15, conforme ilustrado abaixo, no método validateIdentifier.

```
2  public boolean validateIdentifier(String s) {  
3      char achar;  
4      boolean valid_id = false;  
5      if (s.length() > 0) {  
6          achar = s.charAt(0);  
7          valid_id = valid_s(achar);  
8          if (s.length() > 1) {  
9              achar = s.charAt(1);  
10             int i = 1;  
11             while (i < s.length() - 1) {  
12                 achar = s.charAt(i);  
13                 if (!valid_f(achar))  
14                     valid_id = false;  
15                 //i++; //Defeito  
16             }  
17         }  
18     }  
19     if (valid_id && (s.length() >= 1) && (s.length() < 6))  
20         return true;  
21     else  
22         return false;  
23 }
```



## Temporização em Casos de Teste

## Caso de Teste Temporizado (3)

A execução do teste 3 está em looping...

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows a tree view of test cases: IdentifierTestCase - validate03, IdentifierTestCase [Runner: JUnit 4] (containing validate02, validate03, validate04, validate05, validate01), and IdentifierTestCases [validate03]. It indicates 2/5 runs, 0 errors, and 0 failures.
- Identifier.java:** The code for the Identifier class. A red vertical bar highlights the loop from line 15 to line 16. The code includes methods validateIdentifier, validate\_s, and validate\_f.
- IdentifierTestCase.java:** The test case code.
- Outline View:** Shows the class structure with methods validateIdentifier, validate\_s, and validate\_f.
- Problems View:** Shows 103M of 1 problem.



## Temporização em Casos de Teste

## Caso de Teste Temporizado (4)

Considere que a classe de teste fosse alterada como ilustrado abaixo.

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Before;
4 import org.junit.Test;
5
6 public class IdentifierTestCase {
7
8     private Identifier id;
9     public static final int LIMIT=200;
10
11     ...
12
13     @Test(timeout=LIMIT)
14     public void validate03() {
15         boolean obtido;
16         obtido = id.validateIdentifier("A1b2C3d");
17         assertEquals(false, obtido);
18     }
19
20     @Test(timeout=LIMIT)
21     public void validate04() {
22         boolean obtido;
```



## Temporização em Casos de Teste

## Caso de Teste Temporizado (5)

Após o timeout a execução termina acusando problemas na execução dos testes...

```

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
public class IdentifierTestCase {
    private Identifier id;
    public static final int LIMIT=200;
    @Before
    public void initialize() {
        id = new Identifier();
    }
    @Test(timeout=LIMIT)
    public void validate01() {
        boolean obtido;
        obtido = id.validateIdentifier("a1");
        assertEquals(true, obtido);
    }
    @Test(timeout=LIMIT)
    public void validate02() {
        boolean obtido;
    }
}

```



## Temporização em Casos de Teste

## Caso de Teste Temporizado (6)

Após o correção do defeito, os testes voltam a executar normalmente...

The screenshot shows the Eclipse IDE interface with the Java perspective. The central area displays the JUnit Test View, which shows the execution results of the IdentifierTest.java test cases. The results indicate 5 successful runs, 0 errors, and 0 failures, completed in 0.012 seconds. Below the results, the Failure Trace section is empty. To the right of the test results, the IdentifierTest.java source code is shown in the Editor view. The code defines a Identifier class with a validateIdentifier method that iterates through the characters of a string to validate them. The IdentifierTest class contains five test methods: validate01, validate02, validate03, validate04, and validate05, all of which pass. The Outline view on the far right shows the structure of the Identifier class, including its methods.

```

Java - Identifier/src/main/java/Identifier.java - ADT
-----
IdentifierTest.java [ ] IdentifierTestCase.java [ ]
Identifier.java [ ] IdentifierTest.java [ ]
1 public class Identifier {
2     public boolean validateIdentifier(String s) {
3         char achar;
4         boolean valid_id = false;
5         if (s.length() > 0) {
6             achar = s.charAt(0);
7             valid_id = valid_s(achar);
8             if (s.length() > 1) {
9                 achar = s.charAt(1);
10                int i = 1;
11                while (i < s.length() - 1) {
12                    achar = s.charAt(i);
13                    if (!valid_f(achar))
14                        valid_id = false;
15                    i++;
16                }
17            }
18        }
19        if (valid_id && (s.length() >= 1) && (s.length() < 6))
20            return true;
21        else
22            return false;
23    }
24
25    public boolean valid_s(char ch) {

```

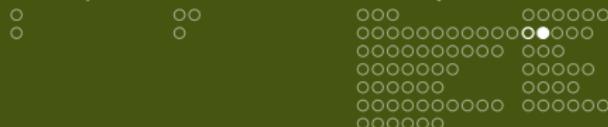


## Teste de Exceções

## Testando Exceções (1)

Embora o exemplo em questão não lance nenhum tipo de exceção, a criação de um caso de teste para avaliar se uma exceção correta está sendo lançada é ilustrado abaixo:

```
51 @Test(expected = IndexOutOfBoundsException.class)
52 public void excecaoString() {
53     String str = new String("Exemplo JUnit");
54     str.substring(30);
55 }
```



## Teste de Exceções

## Testando Exceções (2)

Resultado do teste de exceção: aprovado.

Java - Identifier/src/test/java/IdentifierTestCase.java - ADT

Package Explorer JUnit

Finished after 0,013 seconds

Runs: 6/6 Errors: 0 Failures: 0

Identifier.java \*IdentifierTestCase.java

```
43:     @Test(timeout = LIMIT)
44:     public void validate05() {
45:         boolean obtido;
46:         obtido = id.validateIdentifier("Zm12");
47:         assertEquals(false, obtido);
48:     }
49:
50:     @Test(expected = IndexOutOfBoundsException.class)
51:     public void excecaoString() {
52:         String str = new String("Exemplo JUnit");
53:         str.substring(30);
54:     }
55: }
```

IdentifierTestCase [Runner: JUnit 4] (0,002 s)

- validate01 (0,000 s)
- validate02 (0,000 s)
- validate03 (0,000 s)
- validate04 (0,000 s)
- validate05 (0,001 s)
- excecaoString (0,001 s)

Failure Trace

Outline

IdentifierTestCase

- id:Identifier
- LIMIT:int
- inicializar():void
- validate01():void
- validate02():void
- validate03():void
- validate04():void
- validate05():void
- excecaoString():ve

Problems Javadoc Declaration Console

Writable Smart Insert 67:1 114M of 162



## Definindo um Conjunto de Teste

## Definindo um Conjunto de Teste (1)

- ▶ Um conjunto de teste serve para agrupar várias classes de teste e executá-las em sequência.
- ▶ Na versão 4.x isso é feito por meio de anotações:
  - ▶ `@RunWith` – define qual a classe responsável pela execução dos testes. Quando não especificada, o JUnit utiliza a classe padrão  
`org.junit.internal.runners.TestClassRunner`.
  - ▶ `@Suite` – define o nome das classes de teste que irão compor o conjunto de teste.
- ▶ Para utilizar essas anotações é necessário importar  
`org.junit.runner.RunWith` e  
`org.junit.runners.Suite`, respectivamente.



## Definindo um Conjunto de Teste

## Definindo um Conjunto de Teste (2)

Exemplo de criação de um conjunto de teste  
(TodosTestes.java) incluindo uma única classe:

```
1 import org.junit.runner.RunWith;
2 import org.junit.runners.Suite;
3
4 @RunWith(Suite.class)
5 @Suite.SuiteClasses({
6     IdentifierTestCase.class,
7     // Incluir outras classes de teste aqui.
8 })
9
10 public class TodosTestes {
```



## Definindo um Conjunto de Teste

## Definindo um Conjunto de Teste (3)

A execução de TodosTestes invoca a classe de teste IdentifierTestCase...

```
Java - Identifier/src/test/java/TodosTestes.java - ADT
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
@RunWith(Suite.class)
@Suite.SuiteClasses({
    IdentifierTestCase.class,
})
public class TodosTestes {
}
```

Package Explorer    JUnit    Outline

identifierTestC    ParameterizedTe    TodosTestes.java

Quick Access    Java

Outline

TodosTestes

Package Explorer

Finished after 0,013 seconds

Runs: 6/6 Errors: 0 Failures: 0

TodosTestes [Runner: JUnit 4] (0,001 s)

- IdentifierTestCase (0,001 s)
  - validate01 (0,000 s)
  - validate02 (0,000 s)
  - validate03 (0,000 s)
  - validate04 (0,001 s)
  - validate05 (0,000 s)
  - excecaoString (0,000 s)

Failure Trace

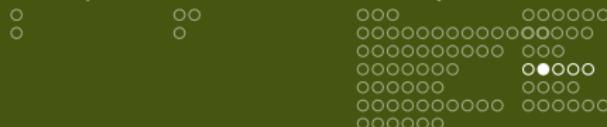
Problems Javadoc Declaration Console



## Executando os Teste sem o Eclipse

## Executando os Teste sem o Eclipse

- ▶ Embora a execução dentro do IDE seja uma facilidade para o desenvolvedor, em algumas ocasiões é melhor a execução fora do ambiente interativo, por exemplo, para a execução via scripts
- ▶ O JUnit oferece mais de uma opção para a execução dos testes fora do IDE, dentre elas:
  - ▶ Via Maven;
  - ▶ Via executor do JUnit – `org.junit.runner.JUnitCore`;
  - ▶ Via executor personalizado – `MyTestRunner`, por exemplo.



Executando os Teste sem o Eclipse

## Execução via Maven

- ▶ Executar o comando **mvn test**, a partir do diretório no qual o arquivo `pom.xml` se encontra.

```
Identifier$ mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building Identifier
[INFO]   task-segment: [test]
[INFO] ------------------------------------------------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] T E S T S
[INFO] ------------------------------------------------------------------------
[INFO] Running IdentifierTestCase
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 0.049 sec
[INFO] Running ParameterizedTestCase
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 sec
[INFO] ------------------------------------------------------------------------
[INFO] Results :
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 1
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
```



## Execução via JUnitCore

- ▶ A partir do diretório no qual o arquivo pom.xml se encontra, execute:

```
Identifier$ java -cp target/Identifier-1.0.jar:target/test-classes:\
> ../../ferramentas/junit4.11/junit-4.11.jar:\
> ../../ferramentas/junit4.11/hamcrest-core-1.3.jar \
> org.junit.runner.JUnitCore TodosTestes
JUnit version 4.11
|.....
Time: 0,011

OK (8 tests)
```

- ▶ O JUnitCore recebe de parâmetro o nome da(s) classe(s) de teste que se deseja executar
  - ▶ Observe, nesse caso, a necessidade de indicar para a máquina virtual Java a localização dos diretórios onde se encontram as classes de aplicação e de teste, além das bibliotecas do JUnit.



## Executando os Teste sem o Eclipse

## Execução via MyTestRunner (1)

- É possível ainda construir um executor personalizado:

```
1 import org.junit.runner.JUnitCore;
2 import org.junit.runner.Result;
3 import org.junit.runner.notification.Failure;
4
5 public class MyTestRunner {
6     public static void main(String[] args) {
7         Result result = JUnitCore.runClasses(TodosTestes.class);
8
9         System.out.println(result.getRunCount() + " tests run on "
10             + result.getRunTime() + "ms");
11        System.out.println("Ignored tests: " + result.getIgnoreCount());
12        System.out.println("Failures detected: " + result.getFailures().size());
13        for (Failure failure : result.getFailures()) {
14            System.out.println(failure.toString());
15        }
16    }
17 }
```



## Executando os Teste sem o Eclipse

## Execução via MyTestRunner (2)

- ▶ A partir do diretório no qual o arquivo pom.xml se encontra, execute:

```
Identifier$ java -cp target/Identifier-1.0.jar:target/test-classes:\n..\..\ferramentas/junit4.11/junit-4.11.jar:\n..\..\ferramentas/junit4.11/hamcrest-core-1.3.jar \\n\nMyTestRunner\n\n8 tests run on 9ms\nIgnored tests: 1\nFailures detected: 0
```

- ▶ Novamente, nesse caso, a necessidade de indicar para a máquina virtual Java a localização dos diretórios onde se encontram as classes da aplicação e de teste, além das bibliotecas do JUnit (junit-4.11.jar e hamcrest-core-1.3.jar).



## Ignorando Casos de Teste

# Ignorando Casos de Teste (1)

- ▶ Existem situações que requerem que algum caso de teste deixe de ser executado:
  - ▶ Caso de teste incompleto.
  - ▶ Caso de teste toma muito tempo da execução.
  - ▶ O recurso exigido pelo caso de teste ainda não se encontra disponível.
  - ▶ O caso de teste está acusando uma falha que ainda não será corrigida.



## Ignorando Casos de Teste

## Ignorando Casos de Teste (2)

- ▶ Remover a anotação @Test do método de teste.
- ▶ Comentar o método de teste.
- ▶ Excluir o método de teste da classe de teste.

Embora funcionem, o executor de casos de teste deixará de reportar a existência de tais testes.



## Ignorando Casos de Teste

## Ignorando Casos de Teste (3)

- ▶ Na versão 4.x é possível simplesmente ignorar um caso de teste.
- ▶ Uso da anotação `@Ignore` ou `@Ignore('Mensagem')`:

```
54 @Ignore
55 @Test(expected = IndexOutOfBoundsException.class)
56 public void excecaoString() {
57     String str = new String("Exemplo JUnit");
58     str.substring(30);
59 }
```



## Ignorando Casos de Teste

## Ignorando Casos de Teste (4)

Saída produzida:

The screenshot shows the Eclipse IDE interface with several open windows:

- Package Explorer:** Shows a tree view of Java packages and classes. A red bar highlights the "TodosTestes" package, which contains the "IdentifierTestCase" class.
- IdentifierTestCase.java:** The code editor window displays the following Java code:

```
44:     @Test(timeout = LIMIT)
45:     public void validate05() {
46:         boolean obtido;
47:         obtido = id.validateIdentifier("Z#12");
48:         assertEquals(false, obtido);
49:     }
50:
51:
52:     @Ignore
53:     @Test(expected = IndexOutOfBoundsException.class)
54:     public void excecaoString() {
55:         String str = new String("Exemplo JUnit");
56:         str.substring(30);
57:     }
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68: }
```

- Outline:** Shows the class structure with methods like validate01 through validate05 and excecaoString.
- Problems:** Shows no errors or warnings.
- Console:** Shows the output "94M of 1".
- Status Bar:** Shows "94M of 1".

In the status bar, it says "Finished after 0,035 seconds" and "Runs: 6/6 (1 ignored) Errors: 0 Failures: 0".





## Testes Parametrizados

## Testes Parametrizados (2)

- ▶ Todos testam o mesmo método `validateIdentifier`.
- ▶ Possuem uma estrutura bastante semelhante.
  - ▶ Variam o parâmetro fornecido ao `validateIdentifier` e o valor esperado.
- ▶ Em situações como essa é possível definir um conjunto de teste parametrizado.
- ▶ Basicamente isso significa:
  - ▶ Criar uma coleção com valores de parâmetros e saídas esperadas.
  - ▶ Definir a estrutura do método a ser testado em função dos valores da coleção.
- ▶ Duas anotações especiais são utilizadas:



## Testes Parametrizados

## Testes Parametrizados (3)

```

1 import static org.junit.Assert.*;
2 import org.junit.Before;
3 import org.junit.Test;
4 import org.junit.runner.RunWith;
5 import org.junit.runners.Parameterized;
6 import org.junit.runners.Parameterized.Parameters;
7
8 import java.util.Arrays;
9 import java.util.Collection;
10
11 @RunWith(Parameterized.class)
12 public class ParameterizedTestCase {
13
14     private Identifier id = new Identifier();
15
16     private String param;
17     private boolean result;
18
19     @Parameters
20     public static Collection<Object[]> data() {
21         return Arrays.asList(new Object[][] {
22             { "Abcd5", true },
23             { "&123", false },
24             { "Inv@lido", false }
25         });
26     }

```

```

29     String param, boolean result) {
30         this.param = param;
31         this.result = result;
32     }
33
34     @Before
35     public void inicializa() {
36         id = new Identifier();
37     }
38
39     @Test(timeout = 200)
40     public void validate() {
41         boolean value =
42             id.validateIdentifier(param);
43         assertEquals(result, value);
44     }
45 }

```



## Testes Parametrizados

## Atualizando TodosTestes (1)

```
1 import org.junit.runner.RunWith;
2 import org.junit.runners.Suite;
3
4 @RunWith(Suite.class)
5 @Suite.SuiteClasses({
6     IdentifierTestCase.class,
7     ParameterizedTestCase.class,
8     // Incluir outras classes de teste aqui.
9 })
10
11 public class TodosTestes {
12 }
```



## Testes Parametrizados

## Atualizando TodosTestes (2)

A execução de TodosTestes invoca as classes de teste IdentifierTestCase e ParameterizedTestCase...

```

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Parameterized;
import org.junit.runners.SuiteClasses;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    IdentifierTestCase.class,
    ParameterizedTestCase.class,
    // Incluir outras classes aqui.
})
public class TodosTestes {
}

```



## Testes Parametrizados

## Outros Recursos para Pesquisar

- ▶ Para mais informações sobre as novidades do JUnit consulte o [Sumário de Mudanças da versão 4.11](#)
- ▶ Algumas delas não abordadas nesse treinamento são:
  - ▶ Definindo a ordem de execução dos testes:  
`@FixMethodOrder` ([mais informação aqui e aqui](#))
  - ▶ Melhorando a identificação de teste parametrizados:  
`@Parameters` ([mais informação aqui e aqui](#))
  - ▶ Declaração de suposições sobre as condições para o teste ter significado: `Assume` ([mais informação aqui](#))
  - ▶ Definição de regras: `@Rule` e `@RuleClass` ([mais informação aqui](#))

Introdução

## Ferramentas Similares



## Ferramentas Similares

- ▶ JUnit não é a única ferramenta de apoio à documentação e execução automática de casos de teste.
- ▶ Considerando as linguagens de programação mais populares, existem outras ferramentas similares, tais como:
  - ▶ TestNG (<http://testng.org/>) para Java.
  - ▶ DUnit (<http://dunit.sourceforge.net/>) para Delphi.
  - ▶ cUnit (<http://sourceforge.net/projects/cut/>) para C.
  - ▶ Jeté (<http://jete.sourceforge.net/>) teste de integração para Java.
  - ▶ Dentre outras...
  - ▶ Uma extensa lista pode ser encontrada em <http://www.testingfaqs.org/> e

Introdução

## Referências



## Referências

- ▶ JUnit Homepage: package, documentation and extensions –  
<http://www.junit.org/>
- ▶ JUnit Start Guide – <http://www.diasparsoftware.com/articles/JUnit/jUnitStarterGuide.html>
- ▶ Harold, E. "An early look at JUnit 4". Artigo On-line, Setembro, 2005. Disponível em: <http://www.ibm.com/developerworks/java/library/j-junit4.html>. Acesso em: 22/11/2006.
- ▶ Gonçaves, A. "Get Acquainted with the New Advanced Features of JUnit 4". Artigo On-line, Julho, 2006. Disponível em: <http://www.devx.com/Java/Article/31983>. Acesso em: 11/22/2006.
- ▶ Open Source Java Developer Testing Tools – Coletânea de endereços para várias ferramentas de teste de código livre (não somente para programas Java) – <http://www.opensourcetesting.org/>
- ▶ Software Testing FAQs – Coletânea de endereços para várias ferramentas de teste – <http://www.testingfaqs.org/>
- ▶ Open Source Testing Tools in Java –  
<http://java-source.net/open-source/testing-tools/>